Chapter 2

Applications

CHAPTER 2. APPLICATIONS

2. APPLICATIONS

The DSP3210 can be used in many different application areas including: telecommunications, speech processing, image processing, graphics, array processors, robotics, studio electronics, instrumentation, and military applications.

ELECTRONIC DATA PROCESSING

■ Mass Memory	Disc controllers, high-precision servo control
■ Workstations	Graphics, translations, rotations, shading, perspective scaling, inversion, multiplication, numeric accelerators, array processing
■ Front-End Processor	Bit manipulation, encryption

INDUSTRIAL

■ Robotics	High-precision servo control
■ Image Processing	Restoration, pattern recognition, compression
■ Process Control	Minicomputer functions
■ Real-Time Simulators	Graphics, servo control, system modeling
■ Instrumentation	Oscilloscopes, FFT, spectrum analysis, signal generators

TELECOMMUNICATIONS

■ PBX	Tone detection, tone generation, MF, DTMF
■ Switches	Tone detection, tone generation, line testing
■ Modems	Echo cancellation, filtering, error correction and detection
■ Transmission	Multi-pulse LPC, ADPCM, transmultiplexing, encryption

GOVERNMENT/MILITARY

■ Sonar	Beam forming, FFT
■ ECM	FFT, adaptive filtering
■ Airframe	Simulation
■ Radar Tracking	Precision FFT, matrix inversions

SPEECH

■ Recognition	Feature extraction, spectrum analysis, pattern matching
■ Synthesis	LPC, format synthesis
■ Coding	ADPCM, LPC, multi-pulse LPC, vector quantization

CONSUMER

■ Studio Electronics	Digital audio
■ Entertainment	High-end video (special effects)
■ Educational	

When the software required for a DSP application is being developed, it is often desirable to use some standard algorithms already in existence. Each application requires a somewhat different use of these algorithms, yet there are great similarities from the processing point of view. Many of the algorithms available are usable in a wide variety of physical problems. The driving force behind most of the algorithms has been an attempt to reduce the computational and data transfer requirements to accommodate the performance constraints of available hardware. Many of these algorithms have been coded in DSP3210 assembly language and are presented in the *AT&T DSP3210 Application Software Library* Reference Manual.

2.1 PC/Workstation Multimedia Applications

The DSP3210 is intended to be used in PC and workstation system architectures in which the DSP3210 is a parallel processor to a host processor. The DSP3210 maintains a 32-bit bus master interface to system memory (see Figure 2-1). The primary benefit of this system architecture is that the DSP is able to access program and data from system memory without host intervention, and expensive local SRAM is replaced by general purpose DRAM. Since the DSP3210 supports both big- and little-endian byte ordering, sharing both data and pointer values with any host microprocessor is easily accomplished. This is especially useful in multimedia applications where intimate communications between the host µP and DSP are necessary. For real-time signal processing, on-chip SRAM is loaded with code and data from system memory before executing. Typically, applications are broken down into functions that are executed successively in this fashion.

This technique is also used to create low-cost EISA and MCA add-on cards. The DSP3210 acts as a bus master on the 32-bit bus to access system memory (rather than adding memory to the DSP3210 card). ISA 16-bit add-on cards typically employ local memory since their 16-bit width would present a memory bottleneck to the DSP3210. However, the visible caching technique employed by the VCOS operating system permits the use of slow, inexpensive DRAM for local memory with minimal impact on system performance. Applications are compatible across all implementations described here.

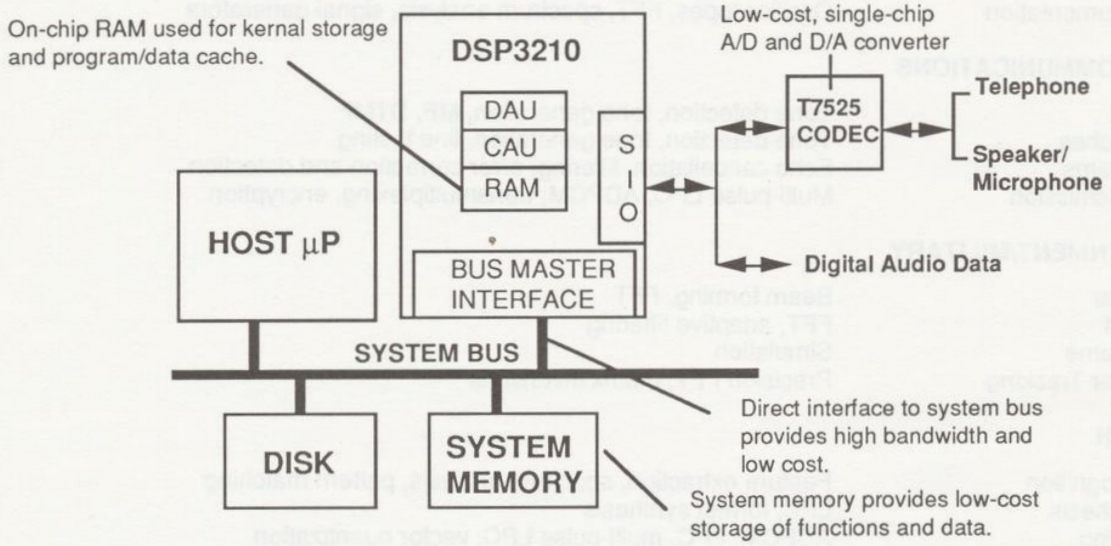


Figure 2-1 Typical PC Multimedia System Configuration

2.2 3-D Graphics Applications

Although the DSP3210 was primarily designed for personal computer/workstation multimedia applications, it is ideally suited to perform the floating-point, compute intensive calculations in a typical 3-D transformation pipeline. In fact, while performing graphics operations, the DSP3210 often outperforms far more costly processors which are touted for their graphics capabilities. These operations consist of matrix multiplies, divides, square root computations, comparisons, and integer conversions. In addition to its native floating-point performance, byte addressability, and large address space, the DSP3210 has on-chip hardware to improve the efficiency of many of these operations such as: reciprocal seed, 8-, 16-, and 32-bit integer to/from floating-point conversion, clip-test register, Z-buffering, etc. A typical graphics system will contain multiple DSP3210 processors that operate in parallel to render the image. The low cost of the DSP3210 makes multiple DSP graphics systems practical. Its page-break and quarter-cycle wait state features significantly reduce system costs while increasing system performance. Inexpensive DRAM can be used for local memory while still achieving the high throughput necessary for high-speed 3-D transformations. The DSP3210's flexible bus protocol also makes it possible for several DSP3210 devices to share the same command and output buffers with little or no performance degradation, further reducing system costs. Figure 2-2 shows a typical 3-D graphics subsystem architecture employing a common command buffer/output buffer configuration.

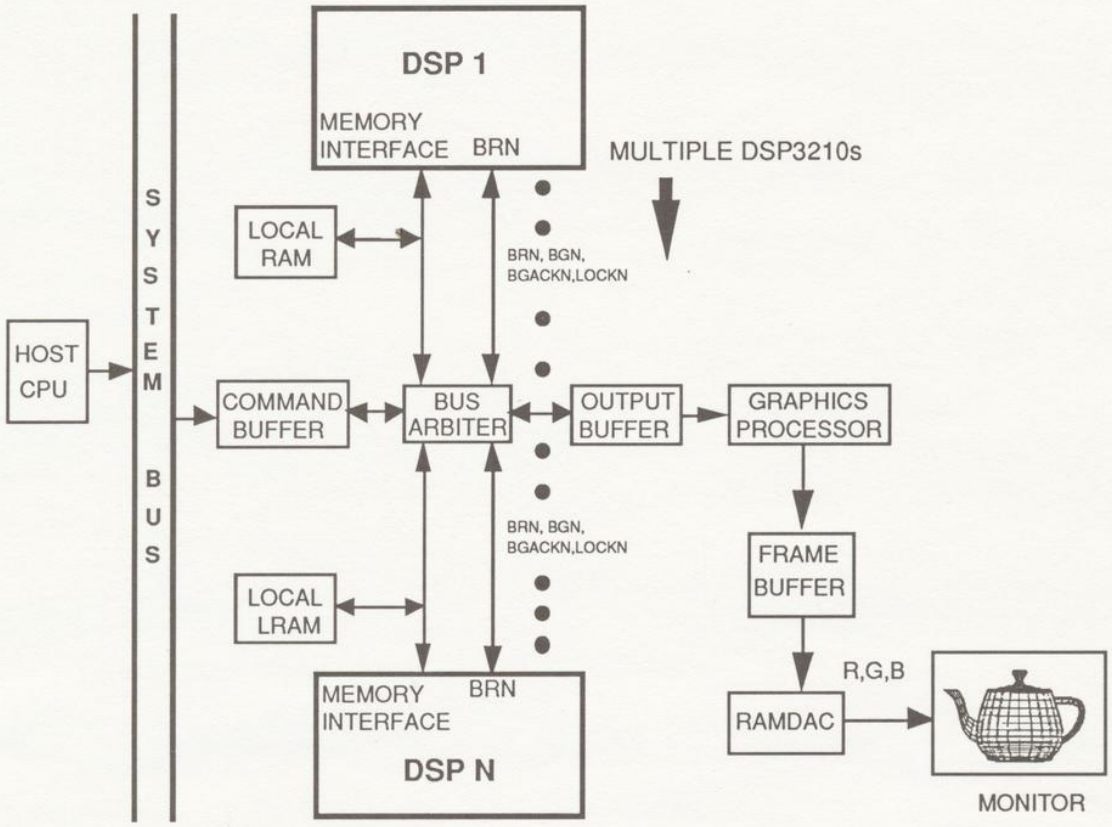


Figure 2-2 Typical 3-D Graphics System Configuration


Smile

Chapter 3

DSP3210 Architecture

CHAPTER 3. DSP3210 ARCHITECTURE

3. DSP3210 ARCHITECTURE

The DSP3210 is a 32-bit, floating-point, programmable digital signal processor. It has been designed for use in numerically intensive applications requiring fast floating-point arithmetic, efficient compiler-generated code, low system cost, and low power dissipation. This has resulted in an architecture with two execution units, a 32-bit integer RISC controller and a 32-bit floating-point data processing unit. Also, two 1K x 32 bit parallel, on-chip memories and a high-bandwidth bus interface deliver operands and instructions to the execution units supporting up to four memory accesses per instruction cycle. On-chip peripherals including a serial interface port, bit I/O port, and timer reduce overall system cost. Table 3-1 summarizes key attributes of the DSP3210 architecture.

Table 3-1 DSP3210 Features/Benefits

Features	Benefits
Full 32-bit floating-point architecture: –Increased precision and dynamic range	Simplifies program development. Provides faster time to market. Much easier algorithm development. Opens up new application possibilities.
All instructions are single-cycle: –No multi-cycle branches	Allows more complex functions or a greater number of simultaneous functions to be performed.
Hardware data format conversions: –IEEE P754 Floating-Point –Integer conversion 8-bit unsigned 16-bit linear 32-bit linear μ-law/A-law	Eliminates lenghty software routines. Permits shared data with host processor or other platforms. Increased throughput in: Graphics and image processing Data communications High quality audio and control applications Telecom and speech applications
Fully vectored interrupt structure with hardware context save	Allows very fast interrupt processing (up to 2.6 million interrupts/sec).
Four memory accesses per instruction cycle	Eliminates memory accessing bottlenecks.
Configurable bus interface: –32-bit addressing –Support for relinquish/retry/bus error –Page break detect	Simplifies/reduces cost of memory controller. Compatible with μprocessor system busses. Improves performance in designs using DRAM.
Byte-addressable address space	Efficient storage of 8- and 16-bit data. Address pointer compatibility with microprocessors.
Flexible wait-state facility: –Each wait-state is 1/4 instruction cycle –Two independently configurable external memory speed partitions	Greater memory speed selection flexibility than conventional full cycle wait-states. Allows mixing of slow and fast memory. Optimizes system speed/cost requirements.
On-chip peripherals: –Serial I/O with DMA capability –32-bit timer	Low cost interface to external devices. Lowers system cost.

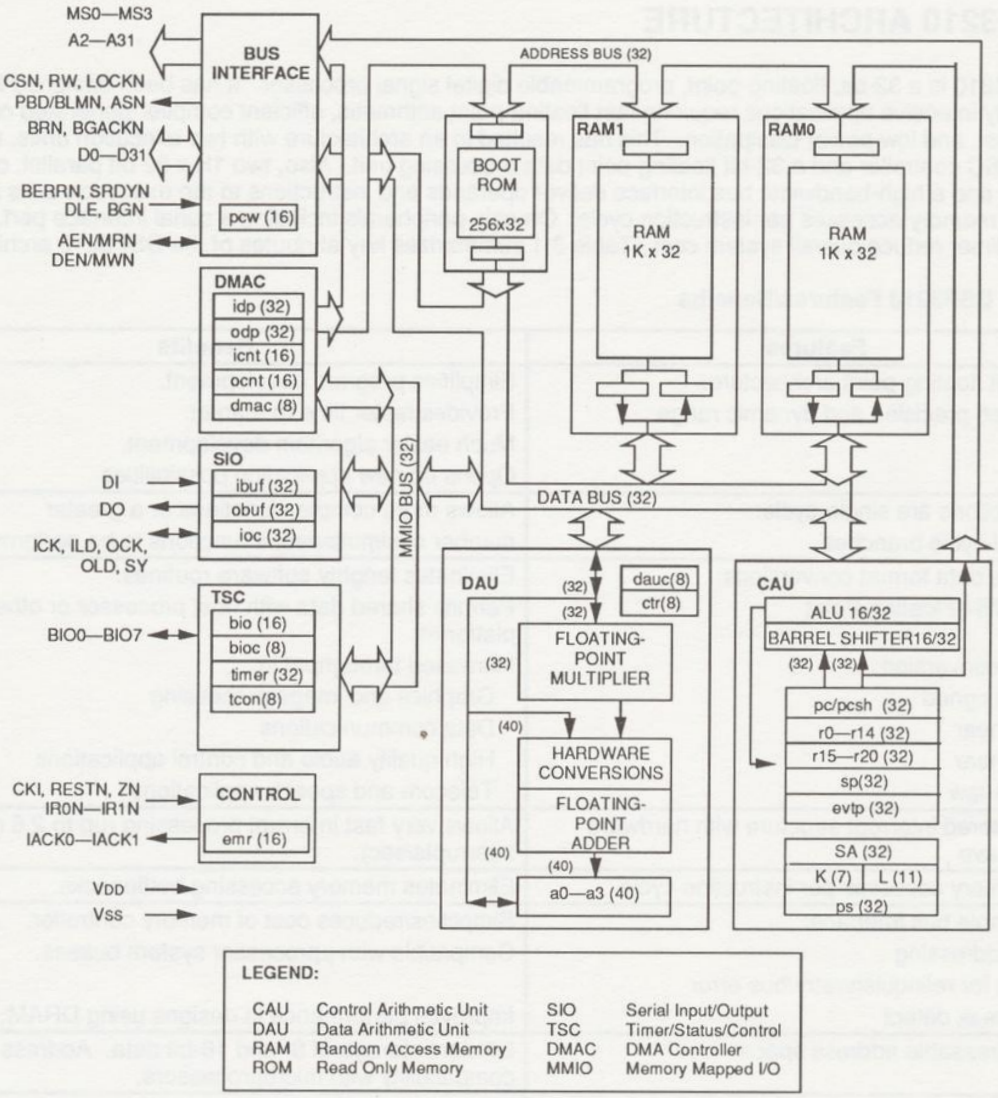


Figure 3-1 DSP3210 Block Diagram

3.1 Functional Units

The DSP3210 consists of seven functional units: Control Arithmetic Unit (CAU), Data Arithmetic Unit (DAU), On-chip Memory (RAM0, RAM1, Boot ROM), Bus Interface, Serial I/O (SIO), DMA Controller (DMAC), and Timer/Status/Control (TSC). Figure 3-1 is a block diagram of the DSP3210.

3.1.1 Control Arithmetic Unit (CAU)

The CAU is responsible for performing address calculations, branching control, and 16- or 32-bit integer arithmetic and logic operations. It is a RISC core consisting of a 32-bit arithmetic logic unit (ALU) that performs the integer arithmetic and logical operations, a 32-bit program counter (PC), and twenty-two 32-bit general-purpose registers. The CAU can execute 16.7 MIPS (Million Instructions Per Second).

The CAU performs two tasks: one executes integer, data move, and control instructions (CA instructions), and the other generates addresses for the operands of floating-point instructions (DA instructions). CA instructions perform load/store, branching control, and 16- and 32-bit integer arithmetic and logical operations. DA instructions can have up to four memory accesses per instruction, and the CAU is responsible for generating these addresses using the post-modified, register-indirect addressing mode—one address in each of the four states of an instruction cycle. For a more detailed description of the CAU, see Section 8.1 Control Arithmetic Unit.

3.1.2 Data Arithmetic Unit (DAU)

The DAU is the primary execution unit for signal processing algorithms. This unit contains a 32-bit floating-point multiplier, a 40-bit floating-point adder, four 40-bit accumulators, and two control registers (dauc and ctr). The multiplier and adder work in parallel to perform 16.7 million computations per second of the form (a=b+c*d). The multiplier and adder each produce one result per instruction cycle. The DAU contains a four stage pipeline: operand load, multiply, accumulate, and result store. Thus, in any instruction cycle, the DAU may be processing four different instructions, each in a different stage of execution.

The DAU supports two floating-point formats, single precision (32-bit) and extended single precision (40-bit). Extended single precision provides 8 additional mantissa guard bits. Post-normalization logic transparently shifts binary points and adjusts exponents to prevent inaccurate rounding of bits when the floating-point numbers are added or multiplied, eliminating concerns like scaling and quantization error. All normalization is done automatically, so the result in the accumulator is fully normalized.

Single instruction, data type conversions are done in hardware in the DAU, reducing overhead required to do these conversions. The DAU performs data type conversions between the DSP3210 32-bit floating-point format and IEEE P754 standard 32-bit floating-point, 16- and 32-bit integer, 8-bit unsigned, u-law and A-law formats. The DAU also provides an instruction to convert a 32-bit floating-point operand to a 3-bit seed value used for reciprocal approximation in division operations. For a more detailed description of the DAU, see Section 8.2 - Data Arithmetic Unit.

3.1.3 On-Chip Memory

The DSP3210 provides on-chip memory for instructions and data. Instructions and data can arbitrarily reside in any location in the on-chip memory. The DSP3210 provides two 1K x 32 RAMs and a 256 x 32 Boot ROM. The memories operate in a parallel fashion to achieve high performance. The Boot ROM is preprogrammed with a routine that can load the internal RAM of the DSP from an external memory, such as an EPROM.

3.1.4 Bus Interface

The external address bus of the DSP3210 is 32-bits wide and fully byte-addressable, allowing the DSP3210 to directly address 4 Gbytes of memory or memory-mapped hardware. External memory is partitioned into two logical address spaces A and B. Each partition contains approximately 2 Gbytes of address space. The number of wait-states for external memory paritions A and B are independently configurable via the pcw register. Configured waits of 0-, 1-, 2-, or 3-or-more wait-states are programmable; this simplifies the interface to fast external memory. Unlike most digital signal processors (which employ full-cycle wait-states), the DSP3210 offers much greater flexibility by offering 1/4 cycle wait-states. Each wait-state is 1/4 of an instruction cycle, allowing much greater granularity in determining optimal speed/cost memory trade-offs. When waits are externally controlled, the DSP adds wait-states until the memory acknowledges the transaction via the SRDYN pin. The bus interface supports retry, relinquish/retry, and bus error exception handling. All signalling provided to the external system is configurable on reset to simplify the interface to a variety of microprocessor system buses.

Sharing the external memory interface is performed via a complete request/acknowledge protocol. System throughput is greatly enhanced by the DSP3210's ability to execute from internal memory while the DSP3210 does not have ownership of the bus. The DSP3210 continues to execute from internal memory until accesses to the external memory are needed. At that point, the DSP asserts the BRN signal and wiats for BGN. The bus arbiter acknowledges the bus request by asserting the DSP3210's bus grant pin, BGN. The DSP3210 then acknowledges the grant by asserting the bus grant acknowledge, BGACKN, and drives the external memory interface pins. When BGN is negated, any on-going external memory transaction is completed before the DSP relinquishes the bus by placing the external memory interface bus in the high-impedance state and negating BGACKN. See Section 6 - Bus Interface.

The DSP3210 shares the external memory interface bus through a request/acknowledge protocol. The external memory interface bus consists of the address bus, A2—A31, the data bus, D0—D31, and associated control pins (See Section 6 - Bus Interface). The DSP is a default bus slave. When the DSP does not have ownership of the bus, it executes from internal memory until accesses to the external memory are needed, and at that point the DSP asserts the BRN and waits for BGN. The bus arbiter acknowledges the bus request by asserting the DSP's bus grant pin, BGN. The DSP acknowledges the grant by asserting the bus grant acknowledge, BGACKN and driving the external memory interface pins. When the BGN is negated, any on-going external memory transaction is completed before the DSP relinquishes the bus by placing the external memory interface bus in the high-impedance state and negating BGACKN. See Section 6 Bus Interface.

3.1.5 Serial I/O (SIO)

The SIO unit provides serial communications and synchronization with external devices. The external signals provided support a direct interface to a time-division-multiplexed (TDM) line, a zero-chip interface to codecs, and direct DSP-to-DSP transfers for multiprocessor applications. The SIO performs serial-to-parallel conversion of input data and parallel-to-serial conversion of output data, at a maximum rate of 25 Mbits/s. It is composed of a serial input port, a serial output port, and on-chip clock generators. Both ports are double buffered so that back-to-back transfers are possible. The SIO is configurable via the ioc register. The input buffer, ibuf, the output buffer, obuf, and the ioc register are accessible as MMIO (Memory-mapped Input/Output) registers in the instruction set.

The data sizes of the serial input and output can be selected independently. Input data lengths of 8, 16, and 32 bits, and output data lengths of 8, 16, 24, and 32 bits can be selected. The input and output data may be selected to be most significant bit first or least significant bit first independently.

SIO transfers can be made under program, interrupt, or DMA control. A program may test the input or output buffer status flags using conditional branch instructions. By configuring the exception mask register, emr, interrupt requests may be generated by the input and output buffer status flags. In DMA mode, transfers occur between ibuf, obuf, and memory without program intervention.

For a detailed description, see Section 9.1 Serial I/O.

3.1.6 DMA Controller (DMAC)

The DMA controller contains two DMA channels, one for input DMA and one for output DMA , that are used in conjunction with the serial I/O. By configuring the input DMA channel, data being shifted into the serial input port

can be buffered in memory without processor intervention. By configuring the output DMA channel, a buffer of data in memory can be supplied to the serial output, as necessary, without processor intervention. The registers to configure the DMA Controller are accessible as MMIO registers in the instruction set. By configuring the exception mask register, emr, interrupt requests may be generated when the memory buffer has been filled or emptied based on the size of the buffer requested.

For a detailed description see Section 9.2 DMA Controller.

3.1.7 Timer

The timer is a programmable 32-bit interval timer/counter that can be used for interval timing, rate generation, event counting, or waveform generation. The input to the timer can be derived from the DSP3210 clock, or it may come from an external source. The output of the timer can generate a maskable interrupt or be selected as an output of the chip to drive external hardware. The count-down timer can be configured to count to zero once, or to count continuously by automatically reloading the counter with its initial value when it reaches zero. The count value may be read or changed at any time during operation. The registers associated with the timer are accessible as MMIO registers in the instruction set. By configuring the exception mask register, emr, interrupt requests may be generated when the count reaches zero.

For a detailed description, see Section 9.3 Timer.

3.1.8 Bit I/O (BIO)

The BIO is a general purpose 8-bit input/output port. It includes features that make it suitable for board-level status signal generation and control signal testing by the DSP3210. The BIO interface consists of 8 I/O lines, which can be independently configured as an input or an output. Outputs can be written with a 1 or a 0, toggled, or left unchanged. Inputs can be directly read and loaded into a CAU register and then tested. The registers associated with the BIO are accessible as MMIO registers in the instruction set. Four of the BIO ports are used on reset to configure the memory map and bus interface. After reset, they can be used arbitrarily.

For a detailed description, see Section 9.4 Bit I/O.

3.2 Processor Control Features

The DSP3210 supports advanced control features that simplify system design and improve software performance. This section overviews serial I/O direct-memory access (DMA), exceptions, and the powerdown mode. A more detailed description of these control functions is presented in Section 7 Processor States.

3.2.1 Serial I/O Direct Memory Access (DMA)

External devices can access the on-chip RAM in the DSP3210, as well as external memory, using DMA. DMA transfers occur between the memory and the serial I/O ports without processor intervention, using cycle-stealing. Two on-chip DMA controllers support memory access via the serial input and serial output ports. See Section 7.1 Serial I/O and Section 7.2 DMA Controller.

3.2.2 Exception Processing

Normal instruction processing can be altered by the introduction of interrupt routines or error handling routines. Exception processing is the set of activities performed by the processor in preparing to execute a handler routine or in returning to the program that took the exception. Error exception and interrupt exceptions cause different activities to be performed. In particular, error exceptions abort the current instruction and can not resume processing. Interrupt exceptions shadow the current state of the processor before taking the interrupt exception; therefore, when the interrupt routine is complete the program can be reinstated and continued.

The error and interrupt exceptions are prioritized and some error sources in addition to all interrupt sources are individually maskable via the exception mask register, emr. A relocatable vector table controls program flow based on the source of the interrupt exception. In response to a given exception, the DSP branches to the corresponding address in the exception vector table which contains pairs of 32-bit words.

The DSP3210 provides a zero-overhead context save of the entire DAU (floating-point unit) for interrupt processing. Interrupt latency is only three instruction cycles for interrupt entry and one instruction cycle for an interrupt exit. Before servicing the interrupt, the DSP3210 automatically saves the state of the machine that is invisible to the programmer, as well as the DAU accumulators a0—a3 (including guards bits), all DAU flag status information, and the dauc register. Internal states that are visible to the programmer are saved and restored by the interrupt service routine. To return to the interrupted program, the interrupt service routine restores the user-visible state of the DSP3210 (that was saved) and then executes the **ireturn** instruction. This single-cycle interrupt return automatically restores the entire DAU state saved during interrupt entry. Quick interrupt entry/exit/context save is critical to real-time multimedia tasks.

For a detailed description of exception processing, see Section 7.4 Instruction Sequencing Exceptions.

3.2.3 Wait-for-Interrupt

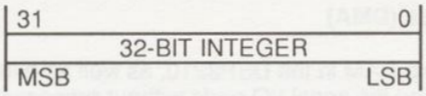
The DSP3210 is equipped with a wait-for-interrupt instruction that stops internal execution in the DAU and CAU. External memory and internal memory operations execute to completion and then wait. Bus arbitration logic remains active in this mode. The peripheral units (Serial I/O, DMA controller, Timer, and BIO) remain active during wait-for-interrupt to perform I/O events and the interrupt handler is active to sense an interrupt request. The DSP will exit the wait-for-interrupt mode when either an unmasked interrupt is requested or an error exception occurs.

3.3 Data Types

Internally, the DSP3210 fully supports three data type formats: 16- and 32-bit 2's complement integers and 32-bit, floating-point numbers. The control arithmetic unit (CAU) performs arithmetic and logical operations on the 16- and 32-bit integer formats. It also has the ability to load and store 8-bit signed and unsigned, and 16-bit unsigned data. The data arithmetic unit (DAU) performs multiply and accumulate operations on the 32-bit, floating-point data and local to the DAU supports a 40-bit extended precision format. The DAU also performs data type conversion between IEEE P754 single precision, 8-bit unsigned, μ-law and A-law, and 16- and 32-bit 2's complement.

3.3.1 32-bit 2's Complement Data Type

The CAU supports a 32-bit, 2's complement integer data type that can be used as 32-bit integer data or for memory pointers. The 32-bit integers range in value from -2^{31} to $2^{31} - 1$. Thirty-two bit arithmetic operations are the default type. All pointers must be 32-bit data types. The organization of 32-bit data in CAU registers is:



3.3.2 16-bit 2's Complement Data Type

The CAU supports a 16-bit, 2's complement integer data type that is used as 16-bit integer data. The 16-bit integers range in value from -2^{15} to $2^{15} - 1$. Because CAU registers are 32-bits wide, when a 16-bit integer operation is performed, data should be loaded into the lower 16-bits of the register (with a sign-extend of bit 15 into the upper 16 bits). A 16-bit operation performs 32-bit arithmetic with flags computed according to 16-bit operation flag rules (see Section 4.1 Flags). If the results are stored back to memory, the lower 16-bits of the result should be stored into a 16-bit memory location. The organization of 16-bit data in CAU registers is:

31 16	15 0
SIGN-EXTENDED	16-BIT INTEGER
	MSB LSB

3.3.3 32-bit Floating-Point Data Type

There are two floating-point formats used inside the DAU: a 32-bit format and a 40-bit format (32-bits, plus 8 mantissa guard bits). The latter format, used only inside the DAU, is not supported by the rest of the device. The DAU accumulators are 40-bits. The guard bits are not user-accessible. For more information about the 40-bit format, see Section 8.2 - Data Aritmetic Unit. The organization of floating-point data in DAU accumulators (a0—a3) is:

39	38 16	15 8	7 0
s	f	g	
	FRACTION	GUARD	EXPONENT
s	MSB LSB	MSB	MSB LSB

The following is a description of the 32-bit floating-point format.

Format

The DSP3210 internal 32-bit floating-point format consists of a 24-bit, normalized, 2's complement mantissa and an 8-bit biased exponent. Figure 3-2 shows the DSP3210 32-bit floating-point format.

31	30 8	7 0
s	f	e
s	FRACTION	EXPONENT
	MSB LSB	MSB LSB

Figure 3-2 DSP3210 Internal 32-bit Floating-Point Format

where: s = the sign bit (0 for positive, 1 for negative)
f = fractional part of the mantissa
s,f = the mantissa
e = the exponent

This floating-point quantity is represented in decimal as follows:

$$N = M * 2^{(e-128)}$$

where: N = the decimal value
e = the exponent, the unsigned quantity "eeeeeeee" expressed in base 10.
M = mantissa, the 2's complement quantity "s(!s).ffff...ffff is equivalent to $(-2)^s$ + 0.ffff...ffff expressed in base 10.
* = multiplication

The magnitude of the mantissa is always normalized to lie between 1 and 2. So, the leading 1 (or leading 0 if a negative mantissa) does not appear explicitly in the floating-point word. This implicit bit is the s bar (!s) that appears above.

Zero

A floating-point value with e=0 is reserved to represent the number zero. Here, the sign and mantissa bits must also be zero. A zero exponent with a nonzero sign and/or mantissa is called a dirty zero, but is treated the same as zero. If a dirty zero is an operand, it is treated as a zero. Any dirty zero generated in the floating-point multiplier or adder is flushed to a valid zero before being written to the accumulators or memory.

Range

The 32-bit floating-point format used in the DSP3210 provides over 1500 dB of dynamic range. The range of nonzero positive floating-point numbers is:

$$N = [1*2^{-127}, [2-2^{-23}] * 2^{127}] \text{ inclusive.}$$

The range of nonzero negative floating-point numbers is:

$$N = [-[1+2^{-23}] * 2^{-127}, -2 * 2^{127}] \text{ inclusive.}$$

The ranges of positive and negative nonzero numbers are almost identical, and the corresponding range for the magnitude of a floating-point number N, in decimal, is approximately:

$$|N| = [5.87747E-39, 3.40282E38]$$

3.4 Memory Organization

The memory space is byte addressable and can support both little endian and big endian byte orderings. The organization of bytes in a 32-bit word is based on the state of the B/LN bit in the pcw register. If B/LN = 0, the little endian mode is selected and byte 0 is the least significant byte of the 32-bit word. If B/LN = 1, the big endian mode is selected and byte 0 is the most significant byte of the 32-bit word. 8-bit data is referred to as a byte, 16-bit data is referred to as a half-word, and 32-bit data is referred to as a word. Each 32-bit word is organized as four bytes, two half-words, or one word. Data must be aligned on the same byte boundary as its operand size. Bytes can be arbitrarily aligned, half words must be aligned on bytes 0 or 2, and words must be aligned on byte 0. Figure 3-3 illustrates the addressing of data for both byte orderings.

Note: $ indicates address

Big Endian (B/LN=1)			
31 23 15 7 0			
Word $00000000			
Half-word $00000000		Half-word $00000002	
Byte $00000000	Byte $00000001	Byte $00000002	Byte $00000003
Word $00000004			
Half-word $00000004		Half-word $00000006	
Byte $00000004	Byte $00000005	Byte $00000006	Byte $00000007
etc.			

Little Endian (B/LN=0)			
31 23 15 7 0			
Word $00000000			
Half-word $00000002		Half-word $00000000	
Byte $00000003	Byte $00000002	Byte $00000001	Byte $00000000
Word $00000004			
Half-word $00000006		Half-word $00000004	
Byte $00000007	Byte $00000006	Byte $00000005	Byte $00000004
etc.			

Figure 3-3 Memory Addressing

The organization of memory in the DSP3210 address space is shown in Figure 3-4. One of the two memory maps is selectable with the C/PN bit of the pcw register. The memory map is divided into seven major sections: Boot ROM, MMIO (Memory-mapped Input/Output), on-chip RAM0, on-chip RAM1, Reserved (for future expansion of ROM and RAM), External Memory A and External Memory B. On-chip SRAM resources consist of two 1K x 32 banks. The Boot ROM consists of 256 words. The MMIO reserves 256 words. The reserved expansion region provides for future expansion of up to 15.5K words of on-chip memory. External Memory A and External Memory B are logical partitions in which a different number of wait-states may be configured for each. The number of wait-states is configured in the pcw register. See Section 6 Bus Interface.

The first address issued on reset is location 0x0 and is located in Boot ROM (if C/PN is initialized to a 1 on reset by BIO7) or external memory (if C/PN is initialized to a 0). The generic code provided in the ROM will allow redirection of the starting address to locations in external memory A or B. Also, the capability to load internal RAM from an external memory device, such as EPROM is provided. This simplifies the use of the DSP3210 in embedded applications. The ROM code is described in Appendix A.

The MMIO memory space reserves 256 words for accessing on-chip memory-mapped IO registers. The DSP3210 contains three peripheral units: SIO, TSC, and DMAC. The control and data registers for these peripherals are addressable in the memory space of the DSP3210 as shown in Figure 3-5 IO Memory Map. Each register occupies one word memory location. For registers smaller than 32-bits, Figure 3-5 shows the location of valid data in the register. MMIO registers are a fixed size and must be read and written using the corresponding size indicator. The address of the MMIO register is dependent on the byte ordering.

Note: Some IO registers have reserved bits for future definition. Those particular bits must be written as zeros for future compatibility.

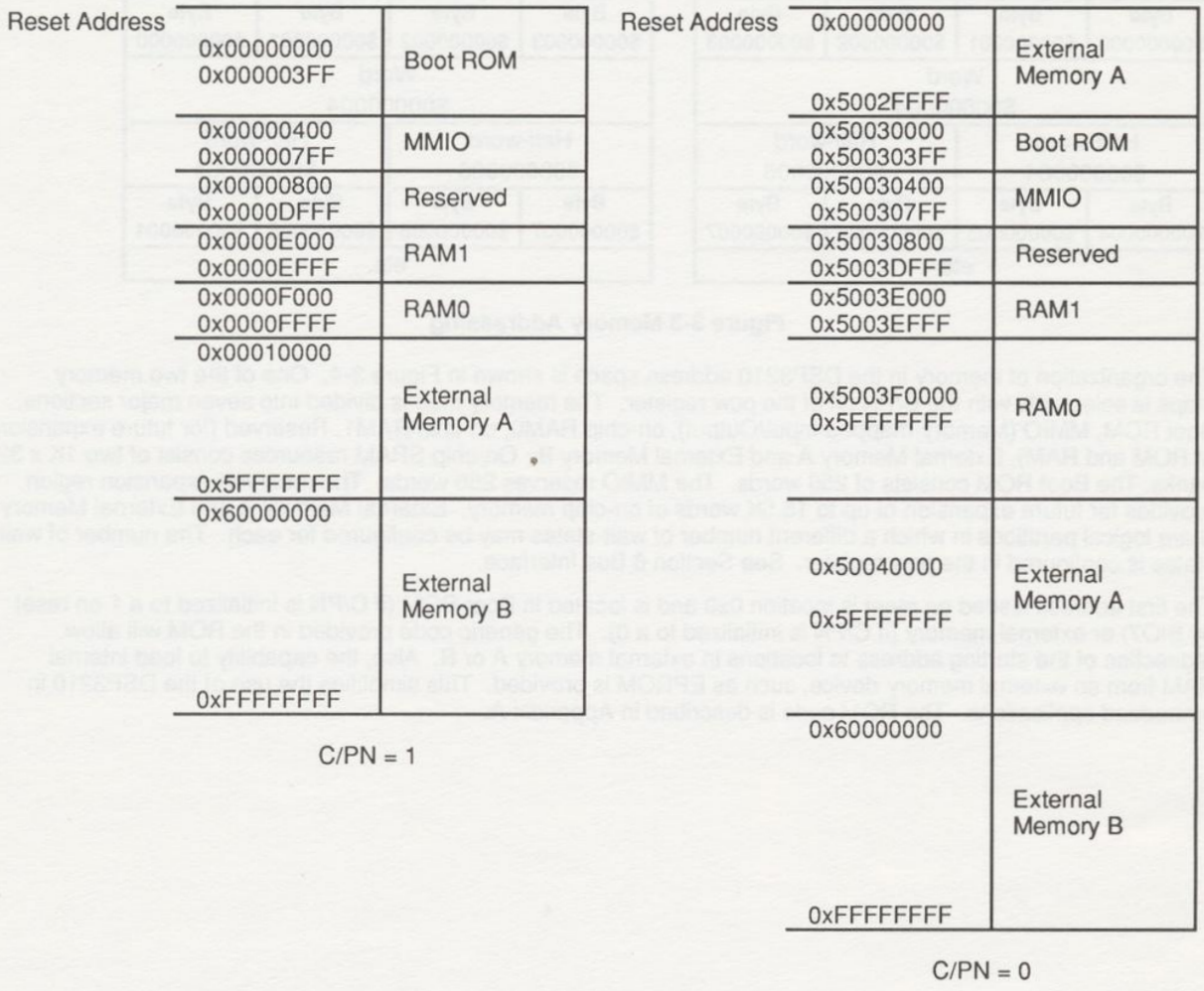


Figure 3-4 Memory Map

16 LSBs of Address (Big Endian)	16 LSBs of Address (Little Endian)	Symbolic Name	Register Bits 31 23 15 7 0	
0x0400	0x0400	Reserved		
0x0404	0x0404	ibuf		R
0x0408	0x0408	obuf		W
0x040C	0x040C	ioc		R/W
0x0413	0x0410	tcon		R/W
0x0414	0x0414	timer		R/W
0x041B	0x0418	bioc		R/W
0x041E	0x041C	bio		R/W
0x0420	0x0420	idp		R/W
0x0424	0x0424	odp		R/W
0x042A	0x0428	icnt		R/W
0x042E	0x042C	ocnt		R/W
0x0433	0x0430	dmac		R/W
0x0434 ... 0x07FC	0x0434 ... 0x07FC	Reserved		

Reserved Bits R/W : Read/Write W : Write Only

Figure 3-5 IO Memory Map

3.5 Addressing Modes

The DSP3210 supports several data access or addressing modes including immediate, memory-direct, register-direct, register-indirect, and register-indirect with postmodification. The addressing modes allowed for each type of instruction are shown in Table 3-2. CA data move instructions, which are used to load CAU registers, support the greatest number of addressing modes and therefore have the most flexibility. The remainder of the instruction types support fewer addressing modes are more tailored to the intended use of the instruction. See Section 4 Instruction Set for more information about each instruction. This section describes the DSP3210 addressing modes.

Table 3-2 Addressing Modes Allowed in Each Instruction Type

Addressing Mode	Instruction Type			
	CA Data Move Group (CAU Reg)	CA Data Move Group (I/O Reg)	CA Arithmetic/ Logic Group	DA M/A and Special Func.
Short Immediate	Yes			
24-bit Immediate	Yes			
Memory Direct	Yes			
CAU Register Direct	Yes	Yes	Yes	
IO Register Direct	Yes			
DAU Register Direct				Yes
Register Indirect	Yes	Yes		Yes
Register Indirect with Postmodification	Yes	Yes		Yes

3.5.1 Short Immediate

Notation: N (16-bit)

In the short immediate addressing mode, the value **N**, a 16-bit 2's complement integer, is supplied by the instruction, and **N** is MSB-extended to 32-bits.

3.5.2 24-bit Immediate

Notation: M (24-bit)

In the 24-bit immediate addressing mode, the value **M**, a 24-bit unsigned integer, is supplied by the instruction, and **M** is zero-extended to 32-bits.

3.5.3 CAU Register (rN) Direct

Notation: rN (N=0-22)

In the CAU register direct mode, the operand is a CAU register specified by the instruction.

3.5.4 IO Register (iorN) Direct

Notation: iorN (N=1-15)

In the I/O register direct mode, the operand is an IO register specified by the instruction.

3.5.5 DAU Register (aN) Direct

Notation: aN (N=0-3)

In the DAU register direct mode, the operand is a DAU register specified by the instruction.

3.5.6 Memory Direct

Notation: *L (16-bit)

In the memory direct addressing mode, the value **L**, a 16-bit unsigned integer, is supplied by the instruction. The effective address is formed such that it always points to the 64K bytes of on-chip memory space. Therefore, when C/PN =1, the upper sixteen bits of the effective address are 0x0000, and when C/PN=1, the upper 16 bits of the effective address are 0x5003.

3.5.7 Register Indirect

Notation: *rP (P=0-22 for CA instructions,
P=1-14 for DA instructions)

In the register indirect addressing mode, the operand is in memory and the address of the operand is in the register specified by the instruction. This register is referred to as a pointer register. The set of registers that can be used for this addressing mode is dependent on the instruction type. For all CA Data Move instructions, CAU registers r0—r22 can be used as pointer registers. For all DA instructions, CAU registers r1—r14 can be used as pointer registers.

3.5.8 Register Indirect with Postmodification

Notation:		
	*rP++	post-increment
	*rP--	post-decrement
	*rP++rI	post-modify by value in register rI (P=0—22 and I=0—22 for CA instructions, P=1—14 and I=15—19 for DA instructions)

In the register indirect addressing mode with postmodification, the operand is in memory and the address of the operand is in the register specified by the instruction, rP. This register is referred to as a pointer register. After the operand address is used, it is modified by ±1, ±2, ±4, or the value in register rI. rI is referred to as an increment register. The post-increment mode modifies the pointer register by +1, +2, or +4 depending on the operand size: byte, half-word, or word. The post-decrement mode modifies the pointer register by –1, –2, or –4 depending on the operand size: byte, half-word, or word. The post-modify by register mode modifies the pointer register by the value in the increment register rI. rI is a 2's complement 32-bit integer. The set of registers, pointers, and increments that can be used for this addressing mode is dependent on the instruction type. For all CA Data Move instructions, CAU registers r0—r22 can be used as pointer registers or increment registers. For all DA instructions, CAU registers r1—r14 can be used as pointer registers, and CAU registers r15—19 can be used as increment registers. If the pointer register is the stack pointer, the operand size is always long word and the post-increment and post-decrement is always by 4 to keep the stack pointer aligned to a word boundary.

3.5.9 Other Addressing Modes

Additional addressing modes, including 32-bit immediate, indexed, relative, memory indirect, bit-reversed, and modulo modes can easily be derived by combining two or more CAU instructions as follows.

—The 32-bit immediate addressing mode is achieved by using the two instruction sequences:

rD = N,
rD = rS <<|N.

—The indexed or base plus displacement mode is achieved by using the instruction **rD = rS + N** and then performing register indirect addressing using rD.

—The relative modes are achieved by using the program counter in the instruction **rD = pc + rS** and then performing register-indirect addressing using rD. rS contains the offset from the pc.

—The memory indirect mode is achieved using the instruction **rD = MEM** and then performing register indirect addressing using rD.

—The bit-reversed addressing mode is achieved by using the carry-reverse add function, **rD = rD # rS**, and then performing register indirect addressing using rD.

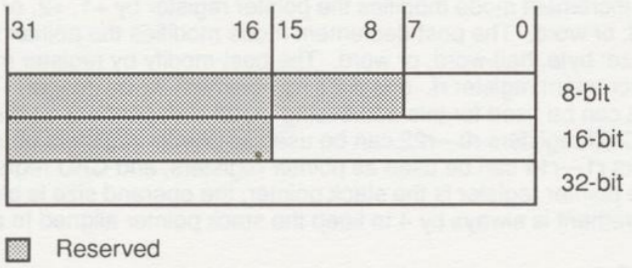
—The modulo addressing mode is achieved by using a two instruction sequence following a postmodification by an increment value. Addresses for the beginning and end locations of the circular buffer are placed in registers.

With:
r1=start_address
r2=end_address
r3=current pointer
r15=increment (step) value
r4=temporary register

Perform postmodification on r3 as usual: **a0=*r3++r15**
Two instruction cycles for modulus operation: **r4=r3-r2**
if(pl) r3=r1+r4

3.6 IO Registers

The IO register set includes control registers and data registers. IO registers vary in size according to their function. IO registers can be 8-bit, 16-bit, or 32-bit registers. The organization of data in the three sizes of registers is:



In general, IO registers are a fixed size and must be read and written using the corresponding size indicator. The IO register set consists of:

IO#	Name	Size
ior0	ps	16-bit
ior8	emr	16-bit
ior12	pcw	16-bit
ior14	dauc	8-bit
ior15	ctr	8-bit

Note: Some IO registers have reserved bits for future definition. Those particular bits must be written as zeros for future compatibility.